Available online at www.sciencedirect.com

SCIENCE DIRECT®

Science of Computer Programming 56 (2005) 275–313

Science of Computer Programming

www.elsevier.com/locate/scico

Quantitative aspects of outsourcing deals☆

C. Verhoef

Department of Mathematics and Computer Science, Free University of Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Received 22 October 2003; received in revised form 15 May 2004; accepted 25 August 2004
Available online 12 October 2004

Abstract

There are many goals for outsourcing information technology: for instance, cost reduction, speed to market, quality improvement, or new business opportunities. Based on our real-world experience in advising organizations with goal-driven outsourcing deals, we identified the most prominent quantitative input needed to close such deals. These comprise what we named *the five executive issues* enabling rational decision making. They concern cost, duration, risk, return, and financing aspects of outsourcing. They add an important quantitative financial/economic dimension to the decision making process. Based on inferred outcomes for the five executive issues, we address the easily overlooked aspects of selecting partners, contracting, monitoring progress, and acceptance and delivery conditions for contracts.

Keywords: Outsourcing; Goalsourcing; Smartsourcing; Fastsourcing; Costsourcing; Offshore outsourcing; Eastsourcing; Tasksourcing; Backsourcing; Insourcing; Scalesourcing; Profitsourcing; Activity-based cost estimation; Total cost of ownership (TCO); Requirements creep risk; Time compression risk; Litigation risk; Failure risk; Overtime risk; Deglubitor risk; Payback period risk

1. Introduction

Many organizations play with the idea to commission their IT activities to third parties. This became known as outsourcing. Reasons to outsource are manifold. For instance,

☆ This research has partially been sponsored by the Dutch Ministry of Economic Affairs via contract SENTER-TSIT3018, project name *CALCE: Computer-aided Life Cycle Enabling of Software Assets.*
E-mail address: x@cs.vu.nl.

0167-6423/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2004.08.003

IT is not the core business, or there is a shortage of IT-developers, or the proper competences are lacking, or in-house development costs are too high. But also union rights can be an obstacle since in some countries they prevent you from making superfluous programmers available to the industry (fire them) once the information technology becomes operational. Or, the organization cannot innovate since all their developers have to maintain the aging legacy portfolio. Or the quality level of in-house developed information technology is becoming unacceptable. Or due to a merger the new board of directors concentrated the IT departments into a new separate organization outside the merged company; sometimes in a joint venture with an existing IT service provider. An example is the joint venture between ATOS Origin and Euronext: AtosEuronext. This was a consequence of the merger of the Belgian, Dutch, and French stock-exchanges who outsourced all their IT-activity to this new joint venture. Needless to say that there are many more forms and reasons for outsourcing. The focus in this paper is the outsourcing of tailor-made applications, so not infrastructure, networks, data centers, etc.

1.1. The in- and outsource cycle

Often the immediate cause of an outsourcing question can be traced back to what is called the *complexity catastrophe* [7]. Business-critical parts of the IT-portfolio became so complex that they are causing all kinds of problems: operational problems, high costs, stopped or stunted innovative power, or other causes. A commonly tried escape from the complexity catastrophe is to abandon the old systems, and start with a clean slate. But then you run the chance of falling prey to the so-called *error catastrophe* [7]. In this error catastrophe, the past is "buried" and all the knowledge that was built-up in the discarded IT-systems is now lost. Sometimes organizations swing from the one extreme of complexity to the other of error-proneness, by throwing their IT-problems over the fence to an IT service provider who promised to clean up the mess. Although sometimes alongside the systems also their support staff is outsourced, the business technologists hardly ever switch jobs in such deals. The risk that the newly found partner is going to solve the wrong problem then becomes realistic. Namely, all the errors that were made along the road that led to the complexity catastrophe are made again. After the problems surface (deadlines missed, wrong functionality, cost overruns, operational disasters, etc), executives realize that outsourcing is not a panacea to solve all their IT-troubles, and a typical reaction is then to backsource the IT. *Backsourcing* (also called *insourcing*) is to bring the outsourced IT back in-house, because it is felt that despite its problems, their IT-assets are better off in-house than outsourced. Then the cycle starts again: sooner or later somehow the complexity catastrophe is entered, and the temptation to throw the problem over the fence grows again.

1.2. Dangerous games

Swinging between the two extremes is not at all productive, and potentially jeopardizes the survival of organizations. In some cases, significant amounts of money are involved in outsourcing deals. This makes them endeavors with a high risk profile. For instance, in [42] we can read that

> The Rolls Royce deal with EDS was worth 45% of its market capitalization, while the Sainsbury's deal with Accenture was 17% of its market capitalization.

Indeed, such deals can directly affect share-holder value. And if the cost, duration, risk, return, and financing of such deals are not properly addressed, it could lead to nose diving stocks, loss of market share, or even bankruptcy if the decisions were taken "on the golf course". Financial news sources contain a flurry of articles testifying to this; we mention just three recent ones:

- The Financial Times reported on March 26, 2003 that the ICI Group had a share price drop of 39% in one day. They attributed this to the failure in its Dutch subsidiary Quest to get the supply chain software to run correctly. Due to persistent late or missed deliveries the largest customers went elsewhere [21].
- The Dutch version of the Financial Times reported on June 20, 2003 that Van Heek-Tweka (textiles) filed for chapter 11 protection. The stock-exchange Euronext suspended trading in this stock. This moratorium on payments was necessary since two weeks earlier, its most important subsidiary went bankrupt. The major cause of the bankruptcy was a failing IT-project. First of all its costs ballooned, and second, this system created havoc in the supply chain [9].
- A local stock market news service reported on June 26, 2003 that Hagemeyer (B2B markets) suspended its implementation of the Global Hagemeyer Solution, for establishing a single ICT-platform for all worldwide activities. Since after the system went live in the UK, one of their most important divisions went from €86 million positive, to 28 million negative. Moreover the market share went from 22% down to 18%. In Australia they will implement this system in a more evolutionary manner, and in the United States they will not implement it at all. The total loss is not disclosed [4].

In [23], a systematic study was carried out where a sample of 150 press announcements of IT-investments was related to the market value of the announcers (59 publicly traded companies). A connection was found between such press announcements and the organization's market value. The cumulative abnormal return over a three-day period around the investment announcement was measured. This return was negative. So these announcements turned out to have a significant negative impact on the market value of the firm [23]. We are not surprised by this, since IT is a production factor in many organizations, so how you deal with IT affects your market value directly. If you announce these investments in the press, they are usually significant, so the underlying information technology is of considerable size. Risk of failure, cost overruns, time overruns, and underdelivery of desired functionality are strongly connected to the size of software (we will see this later on). And since 75% of the organizations have a completely immature software process (CMM level 1) [30, p. 30], almost by definition such investments are exposed to all these risks, resulting in a negative impact on the market value in the long run. Apparently, the investor perceives such announcements not as a value creator, and we think they are right about this. The examples we gave show some of the long-term impacts. Apparently, keen investors do not wait, but react immediately, resulting in a negative impact on the announcer's market value.

Obviously, a more sophisticated strategy is necessary, since gambling with shareholder's capital puts sustainable growth and the continuity of the organization at risk. By now, the shareholder is protected by the Sarbanes-Oxley Act of 2002—an act to protect investors by improving the accuracy and reliability of corporate disclosures [60]. So there are plenty of good reasons why you should get a hold on accurate and reliable data to base your IT-outsourcing decisions on.

1.3. Goalsourcing

Based on the in advance identified goals for outsourcing, a balanced relation with others can emerge. Part of that relation is to ensure that the right responsibilities are taken care of by the appropriate organizations. In particular all stakeholders should understand the long-term consequences of the sometimes far reaching decisions. And a sound quantitative financial/economic analysis is without doubt part of a careful decision process. Due to earlier experienced problems, a mix of in-house work and outsourced activity becomes more and more popular. These mixes are driven by a main goal, hence the name *goalsourcing* (sometimes we see the synonym *tasksourcing*). The idea of goalsourcing is that for a given goal, a mix of activities should be established so that parties perform only those tasks that optimally serve the overall goal.

There are many ways to mix activities: perform activities in-house that you can do fastest, and commission work to others that they can perform faster than you. This mix can be called *fastsourcing*: you maximize to speed-to-market. You can also optimize a mix towards costs: do in-house what is cheapest, and outsource to others what they can do cheaper. This is called *costsourcing*. A commonly used implementation for costsourcing is to contract certain activities to low-wage countries. A popular name for this stems from the US where programming was contracted to low-wage countries offshore of North America. This implementation of costsourcing became known as *offshore outsourcing* (in Western Europe the term *eastsourcing* is used for outsourcing to Eastern Europe). The idea behind an IT-department that is placed outside an organization is to turn an internal cost center into an external profit center: now you can offer your solution to other parties as well. Examples are the just mentioned AtosEuronext, and Sabre. The first offers services to others than the founding fathers of Euronext, and the latter—a joint venture between American Airlines (AA) and IBM—handles the reservations of both AA and other airline companies. The goal that such deals characterizes is to exploit the economies of scale, hence we sometimes refer to it as *scalesourcing*, or the more tantalizing *profitsourcing*. Another mix is to optimize towards quality: business-critical information technology that needs to satisfy particular quality standards. This mix amounts to performing those activities in-house that are done best, and commission to third parties those activities that others excel in. We call this *smartsourcing*.

We like to stress that this paper is not a complete how-to guide for outsourcing issues. Rather it serves as a complement to the aspects and issues that according to our experience are *not* on the radar of decision makers and their supporting staff. We will illustrate our findings via a running example on smartsourcing. The results are applicable to many types of outsourcing deals, and are not restricted to the running example.

1.4. Running example

The author advised several organizations about all kinds of outsourcing deals. This paper brings together the experience gained, and the lessons learned during this field work. For the sake of explanation we composed a running example containing the most prominent quantitative aspects of outsourcing deals. Of course, we modified all the organization-specific data to ensure strict anonymity of the involved organizations. Our running example is a fictitious Federal Government Agency (FGA) that is going to modernize its operations, and is in need for a new management information system supporting its core mission. We call the system CMS, short for Core Mission System. Only very rough requirements and a first sketch in charcoal of the functional specifications are to our avail for the CMS. Of course, there are a number of existing systems that implement parts of the new functionality but there are also new requirements. Despite the rather sketchy shape of our CMS, federal politicians already know the date when the system becomes operational: this is part of the Act that mandated construction of the new CMS. Since sensitive data is going to be processed by this system, the FGA opted for a smartsourcing scenario.

1.5. Becoming a smart buyer

We provide insight into the five executive issues for this running example, so that you can initiate, evaluate, and effectuate you own outsourcing deals by following a similar path. Although the paper is written from the perspective of the problem-owner, both problem-owners and IT-service providers should be in a position to migrate from naive decision makers to realistic and rational negotiators in closing outsourcing deals, after studying our results. Of course, the quantitative data cannot and should not replace the entire rationale for decision making. Many other considerations shape this process, e.g., competitive edge, market share, reputational risk, first/second mover advantage, deepness of your pockets, and so on. The results reported on in this paper focus on a much needed, often neglected dimension that can shed light on the five executive issues: cost duration, risk, return, and financing of outsourcing deals.

1.5.1. Organization of the paper

The rest of this paper is organized as follows. First we discuss how to obtain more information about one of the key indicators we need for our analysis: size information of the information technology under consideration for outsourcing. After we collected such information via various sources and methods, we interpret the information. Based on our best estimate, we address cost aspects, schedule, and financing issues. Then we address the topic of IT-risks both in a quantitative and a qualitative manner. Subsequently, we assess the returns projected by the business, by comparing them to the estimated development and operational costs. With all this information, we can turn our attention to often neglected aspects of selection, contracting, monitoring and delivery. Finally we summarize our conclusions.

2. Collecting size information

First, we need to know more about the amount of IT that is subject to an outsourcing deal. Namely, the amount of functionality is the key from which you can derive the five executive issues. The most reliable [37,36] metric for size is the function point [2,13,25, 15]. At this point you do not need to know what function points are exactly, just think of them as a universal IT-currency converter, giving a synthetic measure of the size of the software. For instance, it takes about 106.7 Cobol statements to construct 1 function point of software. It takes 128 C statements for the same 1 function point [28]. Metrics derived from function points are intuitive in economic analyses. For instance, the cost per function point is comparable to the price per cubic meter for a civil construction. Also this is a synthetic measure, since not all cubic meters are similar, but for economical analysis it is perfect.

To outsource maintenance we need to know how much functionality is being outsourced. This is best measured via source code analysis, with which an as accurate as possible function point count of the existing IT-assets can be conducted. For instance via statement counting and using the language specific factors (106.7, 128, . . .) to recover the function point totals. In case of new development you use the requirements for a function point analysis to obtain an idea of the size [15]. For our running example, the Core Mission System (CMS) of the fictitious Federal Government Agency (FGA), several size estimates were carried out.

2.1. Ball-park estimate

First of all, a fellow federal government agency is asked for advise. Make sure that this friendly estimate is void of commercial bias, and purely technology-driven. Based on similar efforts they already carried out, they came up with a ball-park estimate: between 7500 and 15000 function points. Although this is a very rough idea, it gives information: we are talking a multi-million dollar investment here.

2.2. Measuring the bandwidth

After this first friendly advise, we probed a few outsourcers for an initial cost estimate based on different pricing schemes. One scheme was to see what the minimal size would be, another scheme gave an indication of the maximal size. We used the following pricing schemes:

- Based on your idea of the number of function points, give us a price, and if it is going to cost more, we will pay you additionally for that. For this extra cost, provide us with a price per function point.
- Based on your idea of the number of function points, give us a fixed price, and if it is going to cost more, bad luck for you.

We took the minimum of the answers on the first question, and the maximum of the answers on the second question. This gave us a range between 6000 and 18000 function points. For this estimate it is not necessary to insist on certified function point counting specialists. You are not after the most accurate counting, but assessing potential bandwidth.

2.2.1. Going Scrooge

Note that in some situations you cannot use this method openly: some countries have regulations regarding public offerings that forbid measuring the bandwidth like this. Then you can use shrewd tricks to obtain the bandwidth data. One simple trick is to commission a potential outsourcer to carry out a function point analysis. Often such outsourcers are doing other things for you already, so the next thing you do is to leak secret information to these spies that one of the above pricing strategies is going to be in the public offering. No one is ever going to file you a lawsuit if another pricing strategy is followed later on, since officially they do not know about it. For people feeling a little uncomfortable about manipulative methods, just remember that you are responsible for a multi-million dollar deal and that government regulations are not always the most optimal way to close them. We imagine that Ebenezer Scrooge, the cruel miser created by Charles Dickens in *A Christmas Carol* could have invented this trick.

2.3. Indicative function point count

After these ball-park estimates, an independent certified function point analyst is hired to carry out an indicative function point analysis. The documentation is not yet fit for a detailed function point analysis. After going through the preliminary requirements documents and the functional specifications a number of indicative estimates were made using three indicative methods. One on the basis of data files (5239 FP), one based by weighing the requirements (6700 FP), and an indication based on the functional specifications (7692 FP). The average was taken: 6544 function points. The confidence interval for each method was 50%, and for the average 30% was taken.

For our purpose it is not important what the technical details are that the certified function point analyst used. For decision making it is important whether you can trust the data and what to do and not to do with it.

2.4. Backfiring

Next, we made use of the fact that some functionality of the Core Mission System was available in existing IT-assets within the FGA. Namely, parts of the work process were already implemented using outdated technology marked for retirement after successful implementation of the new CMS. The legacy systems that were identified for replacement were counted using backfiring. A third party was hired to conduct this specialized task. Basically, backfiring is counting the statements using an automated tool, and then using a table with factors turning the statements into function points. If you find 1153 C statements, using the benchmarked C-specific conversion factor of 128, this represents about 9 function points of software. The accuracy of backfiring from logical statements is approximately $\pm 20\%$ [30, p. 79]. The outcome of this source code analysis gave us several totals for statements in several languages, and a total of 6724 function points.

3. Interpreting the collected information

We have collected size information using different sources, different means, and for different purposes. Now we are in a position to review and interpret the data. The ultimate

Table 1
Some indicative data on function point totals and confidence intervals

Method	Min FP	Max FP	Final FP	Confidence (%)
Friendly ball-park	7500	15000	N.A.[a]	N.A.
Enemy ball-park	6000	18000	N.A.	N.A.
Data files	2647	7940	5239	50
Requirements	3350	10050	6700	50
Process model	3846	11538	7692	50
Backfiring	5379	8068	6724	20

[a] N.A. = Not applicable.

goal is to come up with two data points: an indicative internal function point total and a politically correct confidence interval. We summarized all the data points in Table 1.

3.1. Ball-park estimates

The function of both ball-park estimates is to develop an idea of the order of magnitude. For now think of the bandwidth somewhere in between 6000 and 7500 at the minimal side, and between 15000 and 18000 at the high side. No matter what the precise numbers are, one thing is clear: the running example is clearly a major investment, and it is worth the effort to invest in some more involved estimates. But these will cost you money. To give you an idea, for the running example the effort for the function point analyst was about 50 h. The effort for backfiring is measured differently, but think in terms of a few dollar cents per physical line of code.

3.2. Function point estimates

In government situations, it is a good idea to use certified analysts, since in case of major failure, it might come to congressional hearings. And then you at least did everything possible to obtain the best information. Moreover, in the United States, outsourcing deals and their negotiations are subject to a law commonly known as TINA, which is the Truth In Negotiations Act. TINA prescribes certified, calibrated, parametric techniques as a basis for estimating everything necessary for acquisition, including information technology. A complete handbook giving directions is available on the Internet [59]. Also, expect counterchecks in the form of assessments of your plans by external advisors. So there are plenty of reasons to buy the best knowledge available.

No matter how careful you are, also information from certified analysts or certified estimating techniques is not flawless. In our case we found some counting errors, scrutinized proprietary methods used in counterchecks, and cross-examined function point analysts. Reviewing such documents almost always surfaces a few major omissions—that you cannot afford since you base all other estimates on function point totals.

We like to mention an error that is often made, but rarely recognized as such. So this deserves further elaboration. The three different methods that were used were averaged, and the aggregated confidence interval went from 50% to 30%. Although the 50%

confidence intervals were used in the report, we found out during our interview with the specialist that this amount was the numerical representation of his feeling that it was an *unknown but probably large variation*. The problems are summarized:

- you cannot take the average of the three methods;
- but even if you can, the error margin is not decreasing.

Let's see how this works by transposing from the world of software size estimating to the field of length determination for which many of us have more intuition. Suppose we have three methods to estimate the distance between two points.

- A certain muscular tension in your legs represents approximately 1 m. Just walk the distance, you count 1023 steps. Leading to 1023 m.
- Hop in a car, reset the day counter, and drive the distance, look at the day counter. It shows 983 m.
- Use a laser beam, and calculate the distance from the time it takes for the light to travel back and forth. This leads to 981.04 m.

Question: would you take the average of the three methods for the best approximation of the distance? Just the same, it is useless to average the outcomes for the three methods that were used to calculate the size of the Core Mission System. You will take the most accurate one, but since the confidence intervals are in fact unknown, you cannot.

Now the error margin. Suppose for a minute that you can average the three methods, then the error margin is not diminishing. It stays the same: 50%. That is because there is no margin decreasing effect in place. Such an effect can be accounted for when the *same* method is applied repeatedly. For instance, repeat the walk 10000 times, and the error margin will become smaller, and after infinite walks, it will approach the laser beam measurement. But there is no repetitive effect in the function point estimates. So, do not believe averages and diminishing variation unless it is absolutely clear that this makes sense.

3.3. Backfiring estimate

From the backfiring estimate we learned that there are 6724 function points of software in production. One possible way of interpreting this data point is to apply Kim Ross's rule of thumb, which is that the function point count of an upgraded system will be twice that of the old one [53]. This amounts to 13448 function points.

3.4. Horse trading

With all the data points, and their interpretation, the final task is to come to a first estimate that satisfies the following criteria:

- it is justifiable given the investigations,
- it has enough flexibility to manoeuvre within the own organization, when the project scope changes.

This part is not mathematical but political. Depending on the status of a project, the political volatility, and other soft aspects, it is customary to downplay or boost the numbers

that come out of a data collection exercise. For our running example, the final responsible person is the Secretary under which the fictitious Federal Governmental Agency resorts. Since money allocation for such a project cannot easily be changed when new information necessitates this, you have to be careful with the numbers you disclose. In this case it was decided to use a function point total that would give with a 30% confidence interval the maximal total function point count (7692 FP) and round that to one digit significance. Then this was rounded further to numbers with a preliminary feel. The outcome of this political calculation was that the preliminary data to work with for the Core Mission System should be 10K function points ±25%. Of course this information is not broadcasted widely, but used internally to base initial decision making on. This and other estimates are then used to base more involved calculations on. Namely, to infer data for the five executive issues: cost, duration, risk, return, and financing.

3.5. Dealing with arbitrary numbers

Some readers may think at this moment: but what if my estimate is not 10K but some other number, for which no public benchmarks exist? Namely, for different sizes of software the production rates can vary substantially and assignment scopes can differ a little as well. Therefore, you cannot always apply a benchmark for one size to another size. Here we show that if the numbers would have been different, we can obtain the desired answers; this only takes a bit more work. Suppose our best estimate for the CMS is the backfiring result: 6724 function points, plus or minus 20%. Suppose we need to know the average work hours per function point for this size and its confidence interval. There are two possibilities to answer this question:

- Use professional software cost estimation tools.
- Use public benchmarks, and statistical and mathematical techniques.

To answer our question using commercially available tools, we refer you to the vendors. Incidentally, some large organizations use a combination of several commercial tools, in-house developed tools, and statistical/mathematical analysis.

We will show how to answer the above question by doing the math. Although there are no precise data points for 6724 function points in the public domain, we can infer the numbers from public data. In [30, p. 191], we found that for in-house developed MIS systems the average work hours per function point is 4.75 for 100 function point systems, 13.93 for 1000 FP systems, and 38.41 for systems of 10000 function points. We used standard parametric statistical techniques to fit a smooth curve through these three benchmarks. Using an implementation in Splus [61,39] of a nonlinear least squares regression algorithm [8,22,61,46], the three observations can be fitted to the following curve:

$$hfp(f) = 0.6390553 \cdot f^{0.4448014}. \tag{1}$$

In this equation *hfp* is short for hours per function point, and f is the amount of function points. So for a given amount of function points, *hfp* returns the average work hours per function point. This formula is not a perfect fit, but the residual sum of squares is 0.06508417 (zero would have been a perfect fit). Indeed, $hfp(100) = 4.956115$ which is

0.21 off the original data point, $hfp(1000) = 13.802024$, which is 0.15 besides the second observation, and finally $hfp(10000) = 38.436534$, which differs only 2.6% from the original data point. So we can use formula (1) to answer our question: what is the average work hours per function point for an MIS system of 6724 function points, plus or minus 20%? The answer is $hfp(6724) = 32.21609$. For the plus 20% we get: $hfp(8068.8) = 34.93757$, yielding $+8.4476\%$. For the minus 20% we obtain: $hfp(5379.2) = 29.17206$, which is -9.44878%. We used precise numbers here for people who want to check our calculations, but for practical purposes they should be restricted to 3 significant digits. However, you must *not* round the digits in formula (1) (or other statistically fitted formulas), since these numbers approximate the observations as closely as possible. If we round them, the relation is much less accurate.

For completeness sake, we display our interactive dialog with Splus, the program that we used to infer the coefficients of formula (1):

```
% Splus
S-PLUS : Copyright (c) 1988, 2000 MathSoft, Inc.
S : Copyright Lucent Technologies, Inc.
Version 6.0 Release 1 for Sun SPARC, SunOS 5.6 : 2000
Working data will be in /home/x/MySwork
> size <- c( 100,  1000, 10000)
> hfp  <- c(4.75, 13.95, 38.41)
> hfpdata <- data.frame(hfp=hfp, size=size)
> param(hfpdata, "a") <- 1
> param(hfpdata, "b") <- 1
> nls(hfp ~ a * size^b, data = hfpdata)
Residual sum of squares : 0.06508417
parameters:
         a         b
 0.6390553 0.4448014
formula: hfp ~ a * size^b
3 observations
> q()
%
```

We explain this dialog line by line. First we start up the program from a Unix shell (a command line interface). Then there are four lines of boilerplate output, concerning copyrights, and the default location of working data. Then the program is ready for input via its >-prompt. On that line we define a vector `size` containing the three function point sizes (assignment is denoted with `<-`). In the next line we define a vector `hfp` with the three benchmarks for those sizes. Then we make a data frame to create a matrix of the 6 data points. We add in the next two lines parameters `a` and `b` to the data frame, and we give them a value. This value is a guess from our side. Then we use a built-in function `nls` which implements a nonlinear least squares algorithm. We presupposed that the relation between average work hour per function point is a constant around the value of 1 times the size in function points to the power of another constant around 1. As you can see, we need to know in advance what function to expect, and moreover what coefficients to expect. After our incantation, Splus outputs the residual sum of squares, the best possible values for the parameters, the formula for which the fit was carried out, and the amount of observations

on which the relation is based. Since we are satisfied with these results, we quit Splus in the last line, and we are back in the Unix shell.

Summarizing we assume a nice 10000 FP size which eases calculations. But you can deal with other sizes as well, either using math or using commercially available software cost estimation tools that hide the math for you.

4. Cost, duration, and financing

For the calculations that follow we will fix a few parameters (that you can adapt to your situation). We assume 200 working days per year, 8 h of paid working time per day, but only 80% availability of these 8 h/day (which is 6.4 effective hours of working time). Furthermore, we take $80 per hour for internal rates. We also assume from now on to use the 10000 function point measure and the confidence interval of 25% for the running example.

4.1. Different scenarios

Since we are interested in smartsourcing, we have to work out a few scenarios, and decide which one is the best for a particular situation. Basically there are three types of scenarios:

- What if we would do this project in-house? This is the extreme variant without any outside involvement.
- What if we would outsource the project? This is the other extreme: no in-house involvement is present.
- What if we would mix the two above, in other words, if we would smartsource the project? This is the range in between the above two extremes.

4.2. In-house scenario: first cut

Most organizations do not have historical productivity data, but you need such data for making cost calculations. We will use public benchmarks as a surrogate. We start with a first indication of the productivity of in-house developed management information systems, denoted p_i (the subscript i refers to *in-house*). This is the so-called productivity rate: the amount of MIS development one can do in-house per month. We use the following formula taken from [62, p. 61, formula (46)]:

$$p_i(f) = 1.627 + 38.373 \cdot \mathrm{e}^{-0.06222733 f^{0.424459}}. \tag{2}$$

This formula takes an amount of function points, and returns the productivity according to benchmark for that particular size. For a 10000 function point system, this amounts to $p_i(10000) = 3.35$ function points per staff month. But if we know the productivity, we can calculate the cost with the following formula (taken from [62, p. 63, formula (48)]):

$$tcd_i(f) = \frac{rw}{12} \cdot \frac{f}{p_i(f)} \tag{3}$$

Table 2
Activity-based cost estimation for in-house development

Activities	Ascope	Prate	Staff	Effort	Schedule	Cost[a]	%
Requirements	400	73	25	137.5	5.50	1.46	3.7
Initial design	200	81	50	123.7	2.47	1.32	3.3
Detail design	200	61	50	165.0	3.30	1.76	4.4
Coding	150	15	67	687.5	10.31	7.33	18.3
Reuse acquisition	2000	808	5	12.4	2.47	0.13	0.3
Configuration mgt	1500	202	7	49.5	7.42	0.53	1.3
Documentation	1000	61	10	165.0	16.50	1.76	4.4
Unit testing	150	16	67	618.7	9.28	6.60	16.5
Function testing	150	19	67	538.0	8.07	5.74	14.3
System testing	150	20	67	495.0	7.42	5.28	13.2
Acceptance testing	400	28	25	353.6	14.14	3.77	9.4
Project mgt	1000	24	10	412.5	41.25	4.40	11.0
Aggregates	176	2.66	56.94	3758.5	66.00	40.0	100

[a] Cost in millions of dollars.

where $w = 200$ is the number of working days per year and $r = \$640$ the daily rate. For our CMS example this amounts to $tcd_i(10000) = \$31.8$ million.

The duration in calendar months for this project is calculated with yet another formula taken from [62, p. 62, formula (47)]:

$$d_i(f) = \frac{175}{p_i(f)} \tag{4}$$

where the number 175 [30, p. 185, Table 7.4] is the benchmarked assignment scope for in-house MIS development. An assignment scope is the amount of software (measured in function points) that you can assign to one person. This formula takes a number of function points, and returns the schedule in calendar months: $d_i(10000) = 52$ months according to benchmark.

4.3. In-house scenario: activity-based estimate

Now that we have an idea of the productivity and the inferred cost and duration, this first impression justifies that we dig a little deeper, since the costs of this project are in the tens of millions of dollars. To that end we perform an activity-based cost estimation (ABC for short).

When we mix in-house development and outsourced development, we need to decide on a number of activities, and who is going to do them. Most organizations do not have historical data on IT-projects, and have an ad hoc process for each project. So it is likely that it is unknown what kind of major activities an in-house developed MIS project comprise. So again we resort to public benchmarks as a surrogate.

Let us explain Table 2, which is adapted from [30, p.188–9, Table 7.5], that contains such industry averages. The first column defines 12 activities that are commonly carried out during in-house MIS development. The second column contains benchmarked assignment scopes for the given activities: for requirements engineering this is 400, and so on.

You can use it as follows: for a 10000 FP system, it takes $10000/400 = 25$ people to do requirements engineering. The third column contains benchmarked production rates.[1] This reflects the amount of work normally accomplished in a standard time period. The fourth column calculates the amount of staff using the application size in function points and the assignment scopes in column 2. In the fifth column the effort (in months) per activity is calculated. This is being done by dividing the application size in function points by the production rate. The sixth column calculates the schedule in months. Just divide effort by staff, or equivalently assignment scope by production rate. Column 7 calculates the cost per activity in millions of dollars. We did this by multiplying the monthly billing rate by the effort. Column 8 shows the percentage of effort for each activity.

The last row differs from the other rows: it aggregates the column for the entire project. We will explain how to calculate the aggregates. To calculate the overall assignment scope, you cannot just take the average of column 2. An assignment scope is defined as the ratio between size and staff needed for the activity. So we need to know the total number of staff (column 4). For the total number of staff, it is also not a good idea to take the average of column 4. Capers Jones gave the following trivial example to illustrate this issue [32]:

> Suppose we have a small project where 1 person works for 1 month, and then hands the project over to 3 other people who work for 3 months. The schedule totals 4 calendar months and the effort totals 10 months. We have a total of 4 different people involved, but if we divide 10 months of effort by 4 calendar months the effective average staff is only 3.25 people.

Moreover if you take the average of the staff per activity, you end up with an average of 2, which lacks physical semantics. Instead you can approximate total staff by averaging the staff per central activity, while omitting the less effort consuming activities. In this example we took the average of 6 central activities: detailed design, coding, unit testing, functional testing, system testing, and acceptance testing.

Now that we have the aggregate for the average total staff we need for this project, we can calculate the overall assignment scope: it is the total amount of function points, divided by the total staff size, which turns out to be 176. Note that this assignment scope is almost the benchmarked overall assignment scope for MIS development, which is 175 [30, p. 185, Table 7.4]. The overall production rate is calculated by dividing the application size by the total effort. The latter is found by summing the efforts per activity in column 5. The overall schedule is found by dividing the total effort by the total number of staff. The total cost is also a matter of adding up the cost per activity in column 7. The total percentage is found in the same way.

Summarizing, we have an idea of the cost and duration if the project is done completely in-house. Now we turn our attention to the other extreme.

4.4. Outsourced scenario: first cut

Analogously to getting a first impression of productivity, cost and duration of the in-house development, we took some formulas from [62, p. 64, formulas (49–51)] for

[1] We changed these to adapt for our 200 working days.

outsourced development:

$$p_o(f) = 2.63431 + 21.36569 \cdot \mathrm{e}^{-0.01805819 f^{0.5248877}} \tag{5}$$

$$d_o(f) = \frac{165}{p_o(f)} \tag{6}$$

$$tcd_o(f) = \frac{r\,w}{12} \cdot \frac{f}{p_o(f)}. \tag{7}$$

We note that the subscript *o* stands for *outsourced*. The formulas give us a productivity rate of $p_o(10000) = 4.84$ function points per staff month, a schedule of $d_o(10000) = 34$ calendar months, and a total cost of $34.4 million. For the outsourcing fee we used a daily rate of $1000, or $125 per hour. We kept the number of working days per year the same as in-house: $w = 200$.

4.4.1. Offshore outsourcing

It becomes more and more popular to outsource certain kinds of IT activities to low-wage countries. In some cases the hourly fee is only 20% of the internal fees, which comes to $16 an hour. In that case, the total cost of development becomes $4.4 million. Offshore outsourcing has its own overhead costs, such as much higher communication costs, think of private satellite links. In addition the travel expenses will be much higher than for local development. But also substantial additional quality assurance costs are common. Sometimes an offshore team needs extensive training, incurring extra costs as well [38]. So, 30 million is not showing the actual cost savings, in practice this is disappointing, and sometimes even more expensive. To give an idea, a cost saving of 35% was reported on two offshore outsourced reengineering projects, where no full understanding of the software was necessary. Substantial effort was put into improving the internal quality of the software, so that it became more maintainable [11]. You cannot outsource every project offshore, it requires special characteristics in order to be fit for offshore outsourcing. For instance, firm requirements, which was the case in the above reengineering examples. But also clear goals are important, so that communication costs will not balloon, and miscommunication will not put delivery of the right solution at risk. But having very firm requirements in a development project, is like putting the cart before the horses: very precise requirements with rigorous functional specifications are most of the work. Then you can almost use an application generator to write the code. So cost savings is not the best advisor when it comes to offshore outsourcing. If you decide on a cost-only basis, there is a large chance that these lower costs are more than annihilated by the risks.

4.5. An ABC for the outsourced scenario

Now let's do an ABC for the outsourcing case. As in the in-house case we base ourselves on public benchmarks. We note that there is a chance that an outsourcer collected historical data and can deliver internal benchmark data to make more accurate calculations. Of course, the method stays the same, only the actual values differ. We summarized the typical activities that outsourcers use in Table 3 (adapted from [30, p. 270–1, Table 8.6]). There are more activities done by outsourcers than by in-house MIS developers explaining the

Table 3
Activity-based cost estimation for outsourced development

Activities	Ascope	Prate	Staff	Effort	Schedule	Cost[a]	%
Requirements	500	669	20	145.6	7.28	2.43	5.5
Prototyping	2000	101	5	9.9	19.80	0.17	0.38
Architecture	2500	242	4	41.2	10.31	0.69	1.6
Initial design	250	61	40	165.0	4.12	2.75	6.3
Detail design	175	49	57	206.2	3.61	3.44	7.8
Design reviews	175	101	57	99.0	1.73	1.65	3.8
Coding	115	20	87	495.0	5.69	8.25	18.7
Reuse acquisition	2000	808	5	12.4	2.47	0.21	0.5
Code inspections	115	81	87	123.7	1.42	2.06	4.7
Configuration mgt	1500	404	67	24.7	3.71	0.41	0.9
Formal integration	2000	404	5	24.7	4.95	0.41	0.9
Documentation	1000	61	10	165.0	16.50	2.75	6.3
Unit testing	125	36	80	275.0	3.44	4.58	10.4
Function testing	150	48	67	206.2	3.09	3.44	7.8
Integration testing	150	61	67	165.0	2.47	2.75	6.3
System testing	200	81	50	123.7	2.47	2.06	4.7
Acceptance testing	300	121	33	82.5	2.47	1.38	3.1
Project mgt	1200	36	8	275.0	33.00	4.58	10.4
Aggregates	160	3.79	62.49	2640.1	42.25	44.0	100

[a] Cost in millions of dollars.

list of 18 activities. Also in this ABC we used $1000 for the daily rate, and 200 working days per annum.

4.6. An ABC for smartsourcing

In a smartsourcing deal, the problem owner needs to have control over the project, which means that when we look at the ABC for the outsourcing case, we should identify activities that enable control. For instance, if you throw requirements engineering or project management over the fence, are you still in control? The answer is: no. In Table 4 we summarized our ABC for a certain mix between in-house and outsourced activities, that optimizes towards quality of the particular activity. For instance, although outsourcers have an assignment scope of 500 for requirements engineering, and in-house development is 400, we still chose for in-house requirements engineering. The reason is that inside the organization the business logic is known better than outside. As a consequence, the requirements engineering activity takes more staff than if commissioned by an outsourcer. This illustrates that a mix cannot be driven by too many (sometimes conflicting) goals.

4.7. Summary of the estimates

In the previous sections we calculated productivity, cost, and duration of in-house, outsourced and smartsourced development. It will be clear that for the confidence interval of 25%, we can reiterate all the calculations, so that we get a range instead of single values

Table 4
Activity-based cost estimation for smartsourced development

Activities	I/O[a]	Ascope	Prate	Staff	Effort	Schedule	Cost[b]	%
Requirements	I	400	73	25	137.5	5.50	1.47	4.6
Prototyping	O	2000	101	5	9.9	19.80	0.17	0.3
Architecture	I	2500[c]	242	4	41.2	10.31	0.44	1.4
Initial design	I	200	81	50	123.7	2.47	1.32	4.1
Detail design	I	200	61	50	165.0	3.30	1.76	5.5
Design reviews	O	175	101	57	99.0	1.73	1.65	3.3
Coding	O	115	20	87	495.0	5.69	8.25	16.6
Reuse acq.	O	2000	808	5	12.4	2.47	0.21	0.4
Code insp.	O	115	81	87	123.7	1.42	2.06	4.1
Config mgt	I	1500	202	67	49.5	7.42	0.53	1.7
Formal integr.	O	2000	404	5	24.7	4.95	0.41	0.8
Documentation	I	1000	61	10	165.0	16.50	1.76	5.5
Unit testing	O	125	37	80	275.0	3.44	4.58	9.2
Function testing	O	150	48	67	206.2	3.09	3.44	6.9
Integr. testing	O	150	61	67	165.0	2.47	2.75	5.5
System testing	O	200	81	50	123.7	2.47	2.063	4.1
Acc testing	I	400	28	25	353.6	14.14	3.77	11.9
Project mgt	I	1000	24	10	412.5	41.25	4.40	13.8
Aggregates	N.A.	160	3.79	61.94	2982.8	48.16	41.0	100

[a] I = In-house, O = Outsourced.
[b] Cost in millions of dollars.
[c] No in-house benchmark available, we used the outsourced benchmark instead.

Table 5
Summary of key figures that resulted from our calculations

Estimate	Prate	Cost[a]	Schedule	Cost/FP
In-house first cut	3.35	31.8	52	3180
Outsourced first cut	4.84	34.4	34	3440
In-house ABC	2.66	40.0	66	4000
Outsourced ABC	3.79	44.0	42	4400
Smartsourced ABC	3.35	41.0	48	4100

[a] Cost in millions of dollars.

(note that you need mathematical and statistical techniques or commercial software cost estimation tools for that as we explained earlier).

If the investment is going to be substantial, managing the estimates with professional tools and/or with a statistical analyst is paying off. Taking this very seriously will help in addressing issues like counterchecks, litigation and/or congressional hearings, especially in case of major problems. If it then turns out that you used indicative estimation methods only, this could be seen as professional malpractice.

In Table 5, we summarized some key numbers. As can be seen from Table 5, the mix between in-house and outsourced development shows that the costs and schedules of the smartsourcing scenario will be in between the extremes of in-house only or fully

outsourced development. So, it is not as cheap as an in-house development project, and not as expensive as an outsourced project. This is due to the assignment scopes and production rates. If these are different, then the combination can be more effective than one of the extremes: for instance by combining activities in such a way that always the maximal assignment scope is used, or the maximal production rate. Then you can mix for speed to market (fastsourcing), or minimal cost (mostly offshore outsourcing). The smartsourcing scenario is optimized to assure that certain quality standards are met, via taking control, and determining the requirements.

4.8. Financing

For the estimates listed in Table 5 we answer the financing question. Financing means here the costs over time. For that we need to know what the cost allocation over time is for typical IT-projects. One commonly used approximation for that is to assume that effort (this can be cost, or person months) follows a so-called Rayleigh distribution [43–45,47,51,48]. It is of the following form:

$$cad(t) = \frac{ct}{p^2} \cdot \exp - \left(\frac{t^2}{2p^2} \right). \tag{8}$$

In display (8) *cad* is short for cost allocation for development, c is short for cost in dollars, and p represents the month at which the project achieves its peak effort. Using the rule of thumb that half the development time is the peak effort [6], we can plot cost allocation curves for the development of an IT project. Let us plot the cost allocations for the running example. We use the numbers for the three ABCs listed in Table 5.

In Fig. 1 we depicted Eq. (8) for the outsourced, smartsourced, and the in-house scenario for our 10K function point CMS. Indeed, the outsourced scenario is the fastest alternative, but it comes at a somewhat higher cost, the in-house scenario is somewhat less costly, but takes longer, and the smartsourced scenario is in between: not as rapid as outsourced, but not as slow as in-house development. Also with respect to price this alternative is in between: not as costly as the outsourced scenario, but more expensive than full in-house development.

5. Risks

The estimates we have seen so far for cost, duration, and financing are just the beginning. You should see them as *risk-free* estimates, just like the risk-free rate for financial investments. If everything is okay, and there are no risks that could jeopardize the project, these are the estimates to base further negotiations on for your smartsourcing projects. But there are risks, to which we turn our attention now. We will add to the risk-free estimates the dimension of risk. There are many and diverse IT-risks, and we will deal with the most prominent ones.

5.1. Requirements creep

Requirements creep is the risk that *after* the requirements document is finalized, secondary requirements are added, and existing requirements are modified. Suppose they

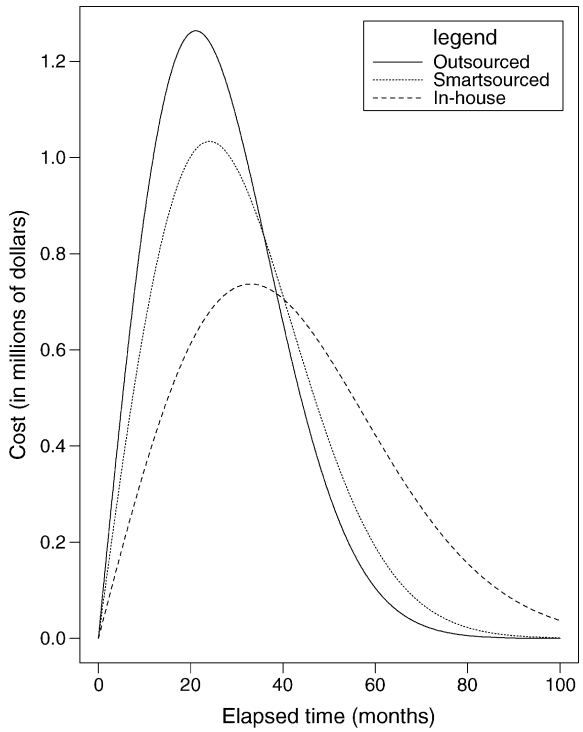

Fig. 1. Visualizing the effort allocation over time for our running example.

would grow with only 1% per month. Let's see how this impacts the amount of delivered function points if the size after the requirements phase is 10000 FP. The creep-adjusted application size is then $10000 \times 1.01^{41} = 15037$ FP, so after finalizing the requirements, the software increases more than 50% in size (we assume 41 months of development after the requirements are set). Of course, this has a major impact on cost, duration, and productivity, up to the point that the system will never be finalized.

Using public benchmarks we can obtain an impression of the requirements creep risk. Table 6 is composed from [29, p. 431, Table 17.4] and [26, Table 4] and shows benchmarked monthly requirements creep rates in several industries. There is also public information on the duration of the growth. We quote from [31]:

> After the requirements are initially analyzed, the volume of new and changing requirements will run between 1% and 3% of the original requirements every month for as long as a year. The total volume of these secondary requirements can exceed 50% of the initially defined requirements at the high end of the spectrum. Creeping requirements in the range of 25% of the volume of original requirements are common.

This gives us enough information to quantify the requirements creep risk for our Core Mission System of the Federal Government Agency. Let us set the growth rate on 2.5%, and

Table 6
Benchmarked monthly requirements creep rates in various industries

Software type	Monthly rate of requirements change (%)
Corporate contract or outsourced software	1.0
Information systems software	1.5
Systems software	2.0
Military software	2.0
Civilian government software	2.5
Commercial software	3.5

Table 7
Software project planned and actual schedules by size of project

Schedule (months)	Project size (function points)			
Average schedule	<100	100–1000	1000–5000	>5000
Planned schedule	6	12	18	24
Actual schedule	6	16	24	36
Difference	0	4	6	12

the growth duration on 12 months. Then for a 10000 function point system we calculate: $10000 \times 1.025^{12} = 13448$ FP. Does this lead to a longer schedule? The answer is yes, but you cannot calculate this with our ABC benchmark calculations. Namely, the production rate is depending on the size of the software system, and they are public for systems of 10000 function points, but not for 13448 FP. So actually we need additional mathematical and statistical methods (or commercial tools), but for an impression we can also use the following trick.

5.1.1. A rule of thumb

We need to obtain a first impression of schedule slips. From Table 7 (taken from [27, p. 4, Table 2]) we can see the planned versus actual schedules by size of project.

This implies that slips are common. And that for systems larger than 5000 FP, this slip is 50% on average. Our ABC estimate for the smartsourcing case for the 10000 FP project was 48 months. The above table suggests that we can estimate the schedule slip at 50% so that the overtime-adjusted schedule becomes $48 \times 1.5 = 72$ months.

5.1.2. Two extremes

So given this high delay, it is worthwhile to dive into different estimates a bit more. Using formulas (2)–(7), we can quantify the differences if it were fully outsourced, or projects done in-house entirely. This is not the real situation, but for obtaining an impression of the risks it will do. We calculate the outcomes of the 6 formulas with the creep-adjusted size of 13448 function points, and then we take the difference with the original values for 10000 function points. We summarize the results in Table 8. This exercise shows a decrease in productivity between 17 and 21%, due to the larger size of

Table 8
Quantifying requirements creep

	Size	Prate	Cost[a]	Schedule	Cost/FP
The in-house situation					
Original estimate	10000	3.35	31.8	52	3180
Creep adjusted	13448	2.76	51.9	63	3859
Difference	3448	0.59	20.1	11	675
Percentages	35%	21%	63%	21%	21%
The outsourced situation					
Original estimate	10000	4.84	34.1	34	3410
Creep adjusted	13448	4.14	54.1	40	4027
Difference	3448	0.70	19.7	6	583
Percentages	35%	17%	57%	17%	17%
The smartsourced situation					
Original estimate	10000	2.49	41.0	48	4102
Creep adjusted	13448	N.A.	55.2	N.A.	N.A.
Difference	3448	N.A.	14.1	N.A.	N.A.
Percentages	35%	N.A.	34%	N.A.	N.A.

[a] Cost in millions of dollars.

the project. It suggests an increase in schedule of the same amount, and a cost increase of 57–63%.

5.1.3. Smartsourced case

The above calculations have the disadvantage that we cannot see the impact on the smartsourcing case. But they gave us insight that most likely there will be schedule delays, productivity decrease and so on. If we conservatively assume that we stay as productive, and can handle still the same amount of function points per activity, we can apply the creep-adjusted size to our smartsourced ABC (displayed in Table 4). After processing the program that implements the ABC with the creep-adjusted size of 13448 FP, we find for the total cost: $55.2 million, which is 14.1 million more than the original, thus an increase of 34%.

5.2. Requirements creep variations

Actually there is a difference between the requirements creep rates of in-house and outsourced development. Recent benchmarks show that in-house requirements creep for MIS projects is 1.2% and for outsourced projects it is 1.1% [31, pp. 186 and 269]. In Table 2, we can see that after 5.50 months, the requirements are finalized, and the total schedule is 66 months, so there is 60.5 months to grow at a monthly rate of 1.2%. This leads to a size increase of 9157 FP, almost 100%. If we look at the outsourced situation, we read from Table 3 that requirements engineering takes 7.28 months, so at a total schedule of 42.25 months, there is almost 35 months of growth. This leads to 4660 additional function points, a little below 50%. It is instructive to see the dramatic differences. If you are a little

more productive, and at the same time manage requirements volatility a little bit better, this has a very large impact on the requirements creep risk.

5.3. Time compression risks

Often a large system comes with a business need, and this need is translated into a deadline. This deadline is often not a result of calculations based upon the set of requirements, but usually based on something else: the next large trade-show, a fantasy deadline like January 1st, or before Christmas, a regulatory date, etc. In the case of our Core Mission System of the Federal Government Agency the reason is political. A law is passed with a time line for implementation, determining the deadline of the CMS as a consequence. It is hard to change this kind of decision making, but sometimes it is necessary to mitigate what we call the time compression risk. Time compression of a software project is trying to do more work in a time frame than you would normally do from a pure technology viewpoint. We can quantify this risk using the following relation between time and effort, which is taken from [50,49]:

$$e \cdot d^{3.721} = constant \tag{9}$$

where e stands for effort, and d is again the duration of a project. This law is not theoretically derived, but rather statistically fit to eight sets of data points containing historical information on comparable IT-development projects. The average value of the power ratio for all eight sets is 3.721. The standard deviation is 0.215. The probability that the true value of the ratio lies between 3.5 and 4.5 is 84%. For more details we refer to [50,49].

Eq. (9) indicates that when we try to *compress* time just a little bit, the *pressure* on the amount of effort increases drastically. This is similar to fluids, where a minimal compression of its volume results in a significant increase of its pressure. Therefore, we sometimes refer to Eq. (9) as the *hydraulic software law*. Let's see what the impact is for our running example. In Table 4 we calculated that the smartsourced variant is going to take 2983.5 person months of effort, and the duration of the project is 48.16 months. With these data points and Eq. (9) we can calculate the *constant* and use it to see what the effect is of varying time. For our running example, the calculated schedule was much longer than the deadline set by the politicians. Therefore, we used Eq. (9) to show the decision makers the consequences of this difference in time.

In Table 9 we quantified the time compression risk in terms of increased effort for a given decrease in duration of the project. As can be seen, trying to do this project in 90% of the schedule, yields an effort increase of 48%, and so on. Assuming that the effort allocation over time of these IT-projects follows a Rayleigh curve, we can visualize the allocation of FTEs over time in Fig. 2. We use a slight variation of Eq. (8):

$$ead(t) = \frac{et}{p^2} \cdot \exp - \left(\frac{t^2}{2p^2} \right) \tag{10}$$

where e is the effort in person-months, *ead* is short for effort allocation for development. The other notations used in Eq. (10) are equal to that of Eq. (8). The solidly displayed curve in Fig. 2 is the same as the dotted smartsourced curve of Fig. 1. This risk-free curve

Table 9
Quantifying time compression risks

Characterization	Duration	Schedule decrease	Effort	Effort increase
Normal duration	48.2	N.A.	2983	N.A.
Slightly compressed	43.3	0.9	4415	1.48
Moderately compressed	38.5	0.8	6843	2.29
Significantly compressed	33.7	0.7	11246	3.77
Heavily compressed	28.9	0.6	19958	6.69
Death march	24.1	0.5	39333	13.19

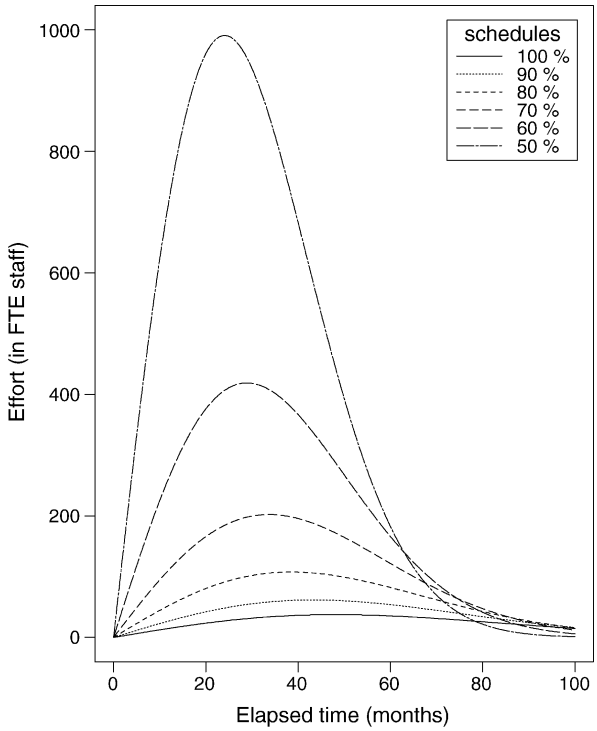

Fig. 2. Visualizing the effort increase for various schedule compressions.

seems almost flat compared to the risk-adjusted curves, where more time compression is reflected in huge increases in effort allocation over time.

The regulatory deadline of our running example was determined by law, and this obliged the schedule to be 26 months instead of 48.16 months, which was estimated for the smartsourcing case. This led to an effort increase of a factor 9.9, which is somewhere between a heavily compressed project and a death march project [64]. Based on the risk-adjusted estimates where we only took the time compression risk into account, the political

deadline was changed to a more realistic deadline, that would not lead to an unacceptable cost-increase due to time compression.

5.4. Failures and challenges

Not every IT-project is finalized in due time, at the estimated cost, giving you the desired functionality. In fact, large software projects fail more often than they succeed. In the initial phase there is not much data so it is hard to quantify the risk of failure, or the risk that the project is going to be seriously challenged. Still, an important early indicator is project size. When the size of a planned software system is substantial, so is the chance on failure and serious cost and time overruns.

In order to obtain an impression for such risks we quantify them by giving a chance of failure for a given size in function points. The following formulas are taken from [62, pp. 45–6, formulas (28) and (30)]:

$$cf_i(f) = 0.4805538 \cdot \left(1 - \exp\left(-0.007488905 \cdot f^{0.587375}\right)\right) \tag{11}$$

$$cf_o(f) = 0.3300779 \cdot \left(1 - \exp\left(-0.003296665 \cdot f^{0.6784296}\right)\right). \tag{12}$$

Formulas (11) and (12) both take a function point total as input and both return a number between zero and one as an output. If you multiply that number by 100% you obtain a percentage indicating the chance of failure according to benchmark. Formula (11) is geared towards in-house development of MIS software systems (hence the subscript i) and formula (12) is based on benchmark data for outsourced development (expressed with the subscript o). Using formulas (11) and (12), we can calculate that for our 10000 function point CMS, the chance of failure when developed in-house is 39% and when outsourced it is 27%. For the mixed case, there is no public benchmark data present, so there is no easy way to quantify the risk of failure as a function of the size of the IT-project. The smartsourced scenario does not need to have a risk between 27 and 39%. For, the additional cross-organizational communication to support the mix could increase the risk outside the range of the two calculated risks. Still, the numbers for the extreme cases give us some indication of the order of magnitude on the risk of failure.

Apart from the chance on failures, there is the chance on overtime projects. Also for this there are two formulas (taken from [62, p. 48, formulas (32) and (34)]):

$$cl_i(f) = 0.3672107 \cdot \left(1 - \exp\left(-0.01527202 \cdot f^{0.5535625}\right)\right) \tag{13}$$

$$cl_o(f) = 0.4018422 \cdot \left(1 - \exp\left(-0.009922029 \cdot f^{0.5657454}\right)\right). \tag{14}$$

For both formulas (13) and (14), we can calculate that the chance on delivering the project much too late is a little over 33.7% for in-house development, and a little under 33.7% for outsourced development. Again, for the mixed case there is no public benchmark data known, so there is no easy formula giving an indication. We will use the numbers we have found as order of magnitude indicators.

Table 10
Approximate distribution of US outsource results after 24 months

Results	Outsource arrangements (%)
Both parties generally satisfied	70
Some dissatisfaction by client or vendor	15
Dissolution of agreement planned	10
Litigation between client and contractor probable	4
Litigation between client and contractor in progress	1

Table 11
Software project outcome by size of project

Project outcome	Project size expressed in function points			
	<100 (%)	100–1000 (%)	1000–5000 (%)	>5000 (%)
Canceled	3	7	13	24
Late by >12 months	1	10	12	18
Late by >6 months	9	24	35	37
Approximately on-time	72	53	37	20
Earlier than expected	15	6	3	1

5.5. Litigation risks

In any outsourcing context, there is a contract and therefore the potential to file a lawsuit in case of unacceptable dissatisfaction by either the offeror or the offeree. For instance if the offeree does not control requirements creep, the offeror is not able to deliver on time. In case of a fixed price contract, it is then possible that some party files a lawsuit. Or, if an offeror bids too low and in the end things turn out to be much more expensive, a dissatisfied client can file a lawsuit. For the CMS this can be done via TINA,[2] if there is a strong suspicion that the offeror was not honest during the negotiations. Often, out of court settlements are the outcome of a litigation conflict, to prevent information on the ins and outs of the problems to become publicly known. Nevertheless there is some information available that gives you an idea in an early phase: before you are closing the smartsourcing deal.

Table 10 is taken from [31, Table 1]. It shows the chance of litigation after 24 months in outsourcing contexts in the United States. For a start, after 24 months about 10% has the plan to dissolve the agreement, 4% deems it probable that litigation is going to take place and in 1% of the cases this is already taking place.

For our running 10000 function point example, we set the chance on litigation at 15% or less. The reasons for this are the alarming numbers we found for failures and challenged projects in the 10K function point range.

Table 11 taken from [26, Table 1] shows a detailed picture of the landscape of challenged and cancelled IT projects where outsourcing played a role. This table clearly

[2] The Truth In Negotiations Act.

shows that for projects larger than 5000 FP, which is the case for our example system, adequate delivery will only occur within 21% of the cases. The cancellation risk is about 24%, and the risk on overtime or very late delivery is substantial: 55% chance on 6 months late, or more. These percentages have the same order of magnitude as the ones we found using formulas (11)–(14).

Analogies also help to obtain an idea of failure and overtime risks. For instance, suppose that our Core Mission System of the Federal Government Agency resembles an IT-project that is known to have failed. Giving the stakeholders a concrete case to ponder on helps to induce an active role with respect to appropriate risk management. Here's a textbook failure that could have happened to our CMS had it been similar to a medical transaction system. We quote a small news flash from the American Hospital Association about the resemblant IT-project [1]:

> Health Care Financing Administration (HCFA) officials got a bipartisan tongue-lashing for wasting $80 million on a computer system that was supposed to improve the accuracy of Medicare payments. As conceived by HCFA, the Medicare Transaction System (MTS) would have consolidated the eight existing Medicare claims-processing systems into a single, national system. HCFA terminated a contract in August with GTE Government Systems to complete the project, after a string of delays and cost overruns. When the contract was signed in 1994, Health and Human Services Secretary Donna Shalala proclaimed that MTS would move Medicare from the "era of the quill pen to the era of the superelectronic highway". American Hospital Association Washington Counsel Mary Grealy found it ironic that the financing administrations defense for the MTS missteps centered on the complexities of the Medicare claims system. For years providers have bemoaned the myriad Medicare reimbursement rules they must follow to avoid allegations of fraud. "Its unfair to claim billing errors as fraud, and the problems with MTS demonstrate the complexities facing providers", she said.

This makes the risks much more vivid than our dry numbers. For an abundance of examples of great failures, computer calamities, and software runaway projects we refer to the books of Robert Glass [18,17,19].

So, using the quantitative data about litigation chances, failure/challenge data from various sources, and comparisons with resembling cases, the FGA got the right mind-set for proactive risk management on their CMS. Not only did we set the litigation chance on 15% max, but also the change on failure on at least 25% if no precautions were taken. These precautions were to dive further into the risks, by way of a workshop, the creation of a risk management plan, and taking preventive measures.

5.6. Risk Assessment Workshop

To obtain a more qualitative view of the risks of a large IT-project, it is worthwhile to organize a Risk Assessment Workshop (RAW). In the RAW the focus should be on the factors that are known to influence the *success* of software development. There are two short lists that can steer the RAW. One list comprises 12 characteristics that successful IT-projects share. They are found by Jones who argues that although there are many and

diverse ways to screw up an IT-project, only a few paths lead to successful software. Successful projects, all share the following 12 essential attributes [27]:

- effective project planning
- effective project cost estimating
- effective project measurements
- effective project milestone tracking
- effective project quality control
- effective project change management
- effective development processes
- effective communications
- capable project managers
- capable technical personnel
- significant use of specialists
- substantial volumes of reusable material

Jones notes that no matter the country, no matter the type of software, the above 12 characteristics tend to be found with successful large software systems in all places where large IT-systems are built. So this list could serve as a yardstick to see how successful the current set up of your IT-project is.

While Jones' characteristics are extracted from successful projects, the other list is based on all kinds of projects (canceled, challenged, and successful). It is the Chaos 10 that stems from Standish Group [56–58]. The Chaos 10 not only comes with a list of subjects but also with a weighted score.

- executive support (18)
- user involvement (16)
- experienced project manager (14)
- clear business objectives (12)
- minimized scope (10)
- standard software infrastructure (8)
- firm basic requirements (6)
- formal methodology (6)
- reliable estimates (5)
- other (5)

The idea is simple: the more points you score, the lower your project risk. Looking into the 12 plus 10 characteristics can further substantiate or disprove the relevant IT-risks. A few were already addressed in a purely quantitative manner. Apart from those types of risk (failure, challenge, time compression, etc), there are other types of risk. We list the most prominent types that should be considered in the RAW:

- technology risk (i.e. the risk of adopting a technology which, for one reason or another, could turn out to be inappropriate)
- synergy risk (the extent to which the project will benefit other projects, business units, or other business)

- alignment risk (the extent to which the project is aligned with the overall mission of the organization)
- organizational risk (the risk that the current organizational structure will be affected by the type of IT system that is to be implemented. This could entail costs which are difficult to foresee at the outset. At the simplest level, this might entail significant staff training, for example.)
- a qualitative view on size risk (we already pointed to the quantitative side of the riskiness of IT projects increasing with the size of the project. Larger projects have a higher risk of not being completed; and larger projects have a higher chance of experiencing cost overruns.)

The outcome of a risk assessment workshop is a list of issues determining the risk-profile of the IT-project. Depending on the seriousness of the resulting risk-profile, the business-criticality of the necessary IT-system, an appropriate strategy towards risk management is developed. This can range from a paragraph on risks in the project plan to a full-blown risk management handbook. Most large business-critical IT projects that contain an outsource component, benefit from a plan on how to backsource the outsourced activities if necessary.

In the case of our example Core Mission System, the RAW confirmed the quantitative findings. The time compression risk was taken care of by a significant deadline extension. The requirements creep risk was harder to address since the regulations were not stable yet, so internal documents were produced to capture the scope of increments in which the system would be constructed. To mitigate this risk, a formal change control board was established, with members from highest management. Furthermore, a sophisticated fallback scenario was planned in case the new system would fail to become operational. Key was to cherish the existing systems as if there were no new initiative so that they could stay in place in case of disaster. This served as a two-edged sword: the geriatric care for the old systems revealed crucial business-knowledge that was needed in the new system as well. But also reusable parts were identified in this manner.

5.6.1. Small risks, small impact?

As you can see, all the risks were seriously addressed. The reason for this is as follows. Although we discussed the risks as separate entities, they often materialize in combinations. For instance, if you have a contract with a firm deadline, and you did not mitigate a small requirements creep risk, your IT-investment will be exposed to a time compression risk that can blow up the project enormously because of "software hydraulics". This in turn will increase the litigation risk, since the software is going to be a lot more expensive than negotiated in the contracts (thus an increase in the risk on challenged projects). Therefore, you should also address IT-risks that you perceive as relatively small on their own, since in combination, they can turn out to be show stoppers.

5.6.2. Deglubitor risk

A maxim contributed to Tiberius goes like this: a good shepherd shears his flock, not flays them. Or in Latin: boni pastoris est tondere pecus non deglubere, hence the ancient name deglubitor for the occupation of a flayer. The better you are aware of the cost, duration, risk, return, and financing aspects of IT-outsourcing deals, the larger the risk

that a subcontractor cannot make a too high price. But this should not lead to a price that is lower than healthy. Because this puts the deal at considerable risk. For, you can turn out to be a bad shepherd, by skinning the subcontractor, leading to a failed project since the subcontractor cannot deliver at a too low price. Alexander Rinnooy Kan (board member and CIO of ING Group) gave an example of this risk during a keynote speech at the IEEE International Conference of Software Maintenance [33]. He told the audience that in the 1990s ING Group made such a good deal with an outsourcer, that the subcontractor requested to dissolve the contract only one year after closing the deal. The reason was that they could not deliver the service at the negotiated price. Obviously, such deals are not in the interest of both parties. So, as a problem-owner seeking an outside party to help you with an IT-investment, practice Tiberius' maxim, and be the shepherd, not the deglubitor.

6. Returns

Calculating potential returns is not a matter of counting function points, filling out a few spread sheets, and voilà you know the return on investment (ROI). There must be a business case for the proposed investment, and thus data on the expected return. We want to find out as soon as possible whether it makes sense to invest or not. For an elaborate treatment of how to quantify the value of IT-investments we refer to [63]. For the sake of simplicity, we assume for our example that the returns are more or less obvious. Namely, in the case of our example system, it was calculated that the new system would save around $333 million. Let's see whether investing makes sense given the development costs, and the future costs while the system is in operation.

It is hard to predict the operational costs while nothing is operational yet, but using a few benchmarks, it is possible to get an idea. The following formula (a modest variation of [62, formula (11)]) is easily derived from a few public benchmarks:

$$mco(f) = \frac{wr}{750} \cdot f^{1.25}. \tag{15}$$

Formula (15) takes the function point total f, and returns the *minimal operational costs* in dollars during the entire operational lifetime (r is the daily burdened rate again, w the number of working days per annum). So the amount of money should be seen as a minimal cost, since no functional enhancements are included in this cost bucket. This formula is based on the life-expectancy benchmark $f^{0.25} = y$ [24, p. 419] and an assignment scope of 750 [28, p. 203] to keep systems up and running. The y stands for operational years. So according to public benchmark, a system of f function points is $f^{0.25}$ years in operation. The annual cost of one person to keep it operational is $r \cdot w$ dollar and we need $f/750$ of them. Multiplication of these factors leads to formula (15). So for our CMS the minimal cost of operation $mco(10000) = \$26$ million (assuming 200 working days per year, and a daily rate of $1000). The development costs were estimated at $41 million, which leads to a minimal TCO of $67 million. Indeed, according to these figures, a little under 40% of this minimal TCO is spent on keeping the system operational. We note that this is indeed rather conservative, since according to many studies [14,5,10,40,52,6,20,28,49,41] carried out over many decades, percentages between 50 and 80% devoted to post-release costs are reported. Let us, next to this conservative estimate, give an estimate at the other end of the

spectrum. Suppose that the development costs form 20% of TCO. Then the remaining 80% amounts to about $165 million, and TCO is about $205 million.

The business case of our example CMS was made prior to its development plan and was estimated to save a third of a billion dollar US, so the $67 million minimal TCO looks then like a viable option, since the net savings are substantial. At the high end, the $205 million leads to savings of more than $125 million over the entire lifetime of the investment, which is 10 years in operation. This still has a positive ROI, but the payback period of this investment is rather long. Let's give an impression of this period: we assume 48.16 months as the investment period (coinciding with the development schedule). We assume that after this investment time our earnings start. Suppose that this is 33.3 million per year (in ten years this is the one-third of a billion dollar). We have to earn the development costs, which is about 41 million, and the operational costs ($165 million in ten years). If this is also spent at an annual rate of 16.5 million, our net income is expected to be 16.8 million per year. It takes 2.44 years to accumulate $41 million. Under these assumptions, the payback period is 6.45 year (2.44 year plus 48.16 months). For simplicity's sake, we did not take monetary inflation, or the cost of capital into account, but this effect leads to an even longer payback period, since dollars tend to deflate. So it is safe to assume a payback period of at least 6.5 years.

If the projected return is lower than in our case, you must account for monetary inflation since in the worst-case scenario the payback period may turn out to be longer than the lifespan of the investment, and/or the net return may turn out to be negative. An important indication of a highly probable underperforming IT-investment is a too long payback period. If it takes many years before an investment generates value, the business has probably changed so much that significant modifications to the original ideas are due before the break-even point. This change induces an additional investment, and potentially vaporizes the originally projected benefits. This is what we call the *payback period risk*.

7. Selection, monitoring, and delivery

In this phase, enough information is present on cost, duration, risk, return and financing of your outsourcing deal to decide whether or not to prepare a Request For Information (RFI) or a Request For Proposal (RFP). If you decide to continue, you still need to gain additional qualitative insight in the proposed IT-investment. Let's suppose that the CMS is a *go*: the budgets are allocated, the deadlines are set to appropriate dates, independent counterchecks did not reveal skeletons in the closet, a plan B is made in case of failure, etc. Now you need to select contractor(s), prepare a contract, and monitor the work once it is in progress. Also, you have to set the criteria under which you agree to accept the delivered information technology. In this section we will address issues that are often overlooked in this phase. We recall that we do not attempt to be complete, but to be complimentary.

7.1. Selection

In addition to the common activities for selecting contractors, it is good to realize that there are a few things at your disposal that can improve the selection process significantly. For a start, you easily fall prey to comparing contractors on a cost-only basis. While cost

is important, it should not be treated separate from, say, open standards, or quality. We deal with open standards and vendor-locking first, and then with quality-driven assessment techniques.

7.1.1. Open standards

Sometimes a contractor comes with a very competitive bid. Then often, the underlying reason is that they already have a lot of technology in place to rapidly deliver the desired functionality. When this is the case, fourth generation languages, domain specific languages, and their application generators usually play a role. The advantage of such tools and techniques is that you can address the problems in a special language, and can deliver function points at high speed. The disadvantage of it, is that the technology is often proprietary, not complying to open standards, and unstable.

- Proprietary means here that the skills necessary for implementation depend on specialized knowledge of a language and tools that only a few people master.
- For such languages and tools, there is a large risk that they are not going to be an ISO standard, or adopted widely by other vendors. When other tools change, most likely other's vendor-specific idiosyncrasies will not be taken into account, which does not happen with open standards.
- Finally, with unstable we mean: requests for change by their current clients are often solved by adapting the domain specific language or the tools, to align even better with the newly discovered business needs. This implies that you have to migrate your code-base to the new version, or that your own additions (via a preprocessor) need to be weeded out, and there is more extra work.

So, if you choose now for a cheaper solution, it can turn out to be an expensive solution later on: when the underlying technology changes, or when you need additional changes and are locked in, so you can no longer negotiate a fair price when the license date is due.

7.1.2. Analyze the contractor

Another often overlooked issue is the risk that the contractor is acquired by others. Does the company exist to assist in solving your problems or are the owners trying to get rich via your problems, and hope to be taken over for a good bid by a larger company? Especially, when relatively small companies are doing well using fairly innovative technology, the larger companies see their particular market share stunt. One policy to deal with this, is that the larger company acquires the rising star, but only to retire the promising technology. Thereby, the larger companies secure the investment in their own solution (software, education, people). The customers of the innovator find themselves suddenly in the middle of a legacy crisis. They can be saviored from this crisis by the acquiring company who will be glad to facilitate a migration from the new technology they just retired to their own product. After all, they acquired people understanding the newly retired technology as well, and often they have experienced people knowing how to migrate from another solution to theirs.

A rule of thumb is here that you should only engage in proprietary technology if you can afford to acquire the company yourself if necessary, and if this obviously pays off in the end. A *pay now, or pay later* analysis is necessary when you are seriously considering this

option. In all other cases we recommend not to base your competitive edge on proprietary chips, homegrown operating systems, nonstandard development tools, exotic languages, fancy 4GLs, and other proprietary technology [54]. Developing IT-policy guidelines to understand vendor-locking risks for your information assets is crucial when solutions are critically dependent on purchased information technology.

7.1.3. Qualitative comparisons

The blueprint of an IT-system is its software architecture. It comprises the main components, their interrelations, and the constraints on them. This blueprint can help when you are selecting among the organizations tendering on your RFI or RFP. For instance, if a project is subject to military standards, it should comply with the C4ISR software architecture guidelines [12] (C4ISR stands for Command, Control, Communications, Computers, Intelligence Surveillance, and Reconnaissance). But how do you know that this is being taken care of? How can you be assured that this is part of the actual plans? And, how do you know that this is in line with your own enterprise architecture? For such questions, several tools exist to help you with the answers. We mention a few:

- SAAM: Software Architecture Analysis Method [3, Chapter 9]. SAAM is used to analyze a software architecture to check whether it satisfies certain properties. You can use this to analyze whether the selected contractor is going to address the important quality attributes of your IT-investment. For instance, maintainability, portability, modularity, reusability, reliability, performance, security, viral protection, usability, privacy, etc. In addition, when business goals change, you can obtain insight about the impact it has on those quality attributes via the software architecture. SAAM assesses which software architecture is most likely to address your needs for the operational period.
- ATAM: Architecture Trade-off Analysis Method [35]. Of course, some of the quality attributes will conflict: reliability will affect performance, security affects maintainability, and so on. ATAM helps you to prioritize the quality attributes for your IT-investment, and it can be used to verify whether the contractor is going to set the same priorities.
- CBAM: Cost-Benefit Analysis Method [34]. Not only after the requirements are completely set, but also beforehand, the stakeholders will want to include many and diverse features in their IT-investment. With CBAM you gain insight in how much of those are realistic, given the budget. You can use CBAM to figure out whether the contractor is planning to implement such unrealistic features.
- SACAM: Software Architecture Comparison Analysis Method [55]. SACAM is a qualitative method to compare software architectures of different candidates among each other, and to compare your own software architecture with that of a candidate. It will help during the selection process to add the dimension of quality to the often cost-only comparisons.

Of course, you cannot ask others to tender on an IT-investment if you have no idea at all about the functionality you wish to outsource. Therefore, it is recommended to utilize the above tools also during the process of shaping the ideas. When the time has come to

select among promising candidates, you can use the tools again but then in the outsourcing context, and SACAM in particular is designed with that in mind.

7.2. Contracting and monitoring

Suppose you made a selection, then the next thing you need is a contract. This contract should contain quantitative aspects that can be objectively measured for monitoring progress, and to base delivery criteria on.

Therefore, you need to collect metric information during and after delivery. Intuitively, it is clear that during development the number of pre-delivery defects should decrease the more we enter the delivery date. Or, the test-case volume should give an idea of how much functionality is being implemented, how fast it is implemented, and whether the testing phase is taken serious at all. The spending rate provides insight whether or not this conforms to the planned spending patterns. For instance, if there is a higher spending rate in the early phases, this is a signal that the estimated total investment cost is too low [16]. Intermediate function point counts provide insight in issues like the severity of requirements creep, but also consolidate earlier indicative estimates. We experienced problematic situations where, after a first estimate, no additional estimations were performed, causing unpleasant surprises in the end. Most contracts do not contain explicit clauses on defect tracking, requirements creep, test-case volumes, intermediate size analyses, etc. We recall that this is not a complete guide to contract management, but we discuss some often overlooked aspects of it.

Of course, the rigor of a contract depends on the five executive issues: if low cost, and no risk is at stake, there is no need for top heavy contracts. In our running example the CMS we are dealing with a significant investment pertaining risks that need to be addressed in the contract.

Historic information characterizing success and failure of outsourcing projects can help. Although publicly accessible information on this topic is rather scarce, Capers Jones published important patterns of (un)successful projects [27,31]. In the 10000 function point range, he came up with the following quantitative data. Successful outsourcing deals are characterized by the following values:

- less than 1% monthly requirements changes after the requirements phase
- less than 5.0 defects per function point in total volume
- more than 65% defect removal efficiency before testing begins
- more than 94% defect removal efficiency before delivery

The *defect removal efficiency* is the percentage of software errors that is found and removed before delivery. Unsuccessful projects in the 10000 function point class usually are characterized by different values for the same metrics:

- more than 2% monthly requirements change after the requirements phase
- more than 6.0 defects per function point in total volume
- less than 35% defect removal efficiency before testing begins
- less than 85% defect removal efficiency before delivery

Of course, it takes experience and effort to measure these indicators, and it does not mean that you can forget about other methods to track progress and to trace potential problems. But if this kind of metric requirement is lacking in your Master Service Level Agreement (MSLA), and is not properly instantiated with actual values in individual statements of work (SOW), there is no objective way to monitor progress, let alone to detect problems in an early phase. Depending on the five executive issues, we recommend a list of measurements like the four by Jones in the MSLA, and that in each individual SOW actual values will be negotiated that can differ per SOW. For instance, if the financial penalties on failing to deliver according to the preset values are substantial, all parties want to prevent that incorrect measurement can occur. For a large SOW this could imply that a trusted third party shall measure the indicators. From Jones' 4 key indicators, the first is straightforward to measure. This amounts to several function point counts, from which the compound monthly growth rate can be calculated: the requirements creep. We devote a separate section to aspects concerning defects, since it is less clear how to deal with them.

7.3. Defects

In order to obtain realistic contracts, it is a good idea to gain insight in the amount of defects that are usually delivered according to benchmarked data. Before we dive into this issue, we give an example outside the software world.

When in the Netherlands a new storm surge barrier was planned by the national government, they required that the barrier should have a fail rate of one in ten thousand. This meant that the barrier may stay open 1 in 10000 cases whereas it should have been closed. Or it closes 1 in 10000 times while it was not necessary. This requirement reflects the feeling that each victim is one too many, but for a contract it boils down to the question whether or not this is a realistic requirement. To detect this, a comparison with the reliability of using the ejection seat among F-16 pilots was made. It turned out that this was in the order of magnitude of one in a thousand. So pilots erroneously eject 1 in 1000 cases, and fail to eject when they should have with the same ratio. This benchmark illustrated that the original requirement had to be replaced by a more realistic one.

In the world of software we see similar unrealistic requirements, like bug-free software, 24/7 uptime, and so on. An availability of 99.9999% for the delivered system is a requirement of which the feasibility is easily assessed. There are 31536000 s in one year, and this availability leads to a down time of 31.536 s/annum. This is less than a minute. Is that with or without software updates? Is that with or without hardware maintenance? Is it really necessary to have this 24/7 uptime? Or is a little less ambitious requirement also in order? For instance, one company required that the intranet site for their ex patriots had a 24/7 uptime. This requirement was altered to a more realistic one, given the business criticality of the software, and the prohibitively high costs of this requirement.

While some of the demands in contracts can be easily corrected using common sense, the aspect of defects is harder to address. The *defect potential* is the sum of errors that potentially occurs in five major categories: the requirements, the design, the source code, the documentation, and in incorrectly repaired errors (this is also called a bad fix). The defect potential in the United States is 5 defects per function point. Table 12, taken from [28, p. 338], shows a distribution of this number over the five major categories, plus

Table 12
US averages for software defect potentials and defect removal efficiency

Defect origins	Defect potentials	Removal efficiency (%)	Delivered defects
Requirements	1.00	77	0.23
Design	1.25	85	0.19
Coding	1.75	95	0.09
Documents	0.60	80	0.12
Bad fixes	0.40	70	0.12
Total	5.00	85	0.75

Table 13
Defect potential for a 10000 function point system and its creep-adjusted size

Defect origins	Defects for 10000 FP	Creep adjusted 13448 FP
Requirements	10000	13448
Design	12500	16810
Coding	17500	23534
Documents	6000	8068
Bad fixes	4000	5379
Total	50000	67240

their removal efficiency. What we see immediately is that there is for each function point, the potential for one requirements error (see Table 13 for our running example). Of course, 77% is found according to Table 12, and they will not all be of the highest severity class. But according to benchmark for a 10K FP system the average number of high-severity defects is 1593 [30, p. 191], which is 15%. For these 15% delivered severe defects very high repair costs are due, plus the damage to business-critical data, or the failure to serve your customers. So it is crucial to know about this potential, and more importantly, how to diminish it.

Alternatively, as a rule of thumb you can estimate the defect potential with the following formula [28, p.196]. For a given system of f function points, the defect potential p is estimated via $f^{1.25} = p$. For our CMS this implies a defect potential of 100000, and 144817 for the creep-adjusted size. Using the average of 85% defect removal efficiency, we find 15000 to 22446 delivered defects.

So it is an idea to put in the contract that a certain defect removal efficiency should be reached. We know that the average defect removal efficiency is about 85% (see Table 12). In order to satisfy predefined levels of defect removal efficiency, an appropriate mix of defect removal and defect prevention methods needs to be applied during development. Examples are inspections of all major deliverables: requirements, designs, code, and test cases. Moreover, a variety of different testing methods can help. As a rule of thumb each separate software testing step finds and removes 30% of the errors [28, p. 198]. Another rule of thumb is that a formal design inspection will find and remove 65% of the bugs present, and each formal code inspection will find and remove 60% of the

present bugs [28, p. 199]. So in order to enforce 95% defect removal efficiency, you can oblige the outsourcer to conduct the mix that should lead to this efficiency. For instance, if you only test, and do not use inspections, about 11 stages of testing are necessary before a cumulative defect removal efficiency above 95% is possible, consuming about 50% of the total development effort [28, p. 557]. If you do use formal design and code inspections, you can achieve such efficiency levels with less impact on the total effort. And you can control that by demanding high pre-test removal efficiency levels.

Suppose you would not take preventive action to address the delivered defect risk. Let's look at the impact for our running example. We zoom in on the latest benchmarks for that. According to [30, p. 191], for a 10000 FP MIS project, the defect potential is 5.90 defects per function point. The defect removal efficiency is 82%. This leads to 10620 defects. According to benchmark, it takes $f/750 = n$ staff members to keep a system of f function points operational. This amounts to 13 to 17 people, using the 10000 and the creep-adjusted 13448 function points. They have to solve the majority of the 10620 delivered defects. Using the simple rule of thumb that a maintenance programmer repairs about 8 post-release defects per month [28, p. 199], they can repair between 1279 and 1721 bugs per year (for the original and creep-adjusted function point totals). Then it takes about 8.3 years to repair the defects (in both cases). Of course, not all errors are repaired, but some are quite expensive, in particular the requirement and design errors. These latent errors will go unnoticed until the operational phase, and some of them need proper repair, with all the consequences.

Summarizing, you need to exploit a contract to enforce defect removal activities, to enforce several testing stages, and to set numerical values on the four key indicators so that the risks are balanced with respect to the investment you are going to engage in. Research showed that for outsourced software that ends up in court litigation because of suspiciously low quality levels, the number of testing stages is around three, and formal inspections are not used at all [28, pp. 549–550]. Performing 3 testing stages accumulates to an aggregate defect removal efficiency of about 70% [28, p. 557]. Translated in contract language, such levels are almost an assurance for litigation conflicts.

7.4. Delivery

Suppose that you dealt with the contracting and monitoring issues. The metrics are now an integral part of the contracts. By monitoring the indicators the foundation to base the acceptance criteria of the end-product on are firmly grounded. A mistake that is often made, is to base the delivery conditions on a so-called production-acceptance test: if it passes the tests when it is installed in the production environment, we accept it. Table 14, taken from [31, Table 9], shows why this is a mistake: only between 25 and 35% of the defects are detected during acceptance testing. So it is not the proper criterion to base acceptance of the delivered information technology on.

But since you are now well prepared, you will not make such errors anymore. Instead you took care of these risks in the contracting phase, and assessed the key indicators while development was in progress, so that unpleasant surprises cannot spoil smooth delivery. Of course, these quantitative aspects of delivery criteria are no replacement for proper management, they just add to it.

Table 14
Software defect removal efficiency ranges

Defect removal activity	Ranges of defect efficiency (%)
Informal design reviews	25–40
Formal design inspections	45–65
Informal code reviews	20–35
Formal code inspections	45–70
Unit test	15–50
New function test	20–35
Regression test	15–30
Integration test	25–40
Performance test	20–40
System test	25–55
Acceptance test (1 client)	25–35
Low-volume beta test (<10 clients)	25–40
High-volume beta test (>1000 clients)	60–85

8. Conclusions

In this paper we have given a set of formulas, rules, guidelines, tricks, and rules of thumb, to come to grips with what we call the five executive issues. Namely, the costs of IT systems, the duration of their construction, the risks involved, the potential returns, and their financing over time. We discussed these quantitative aspects in the context of closing outsourcing deals for information technology. We identified several types of outsourcing, all driven by specific goals, such as optimizing to speed, cost, quality, control, economies of scale, and so on. To illustrate the closing of outsourcing deals, we treated a running example based on our real-world experience with such deals optimized to quality. For this smartsourcing example we dealt with the five executive issues. Based on the outcomes, we illustrated how to proceed with the quantitative side of the selection process, contract management, monitoring progress, and delivery criteria. The material in this paper serves to add a quantitative dimension to the already complex problem of making the business case for outsourcing. And as a final piece of advise, whatever the outsourcing occasion: never outsource your brains.

References

[1] AHA News, HCFA in hot seat, October 6, 1997. 2002 version available via, http://www.ahanews.com.
[2] A.J. Albrecht, Measuring application development productivity, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, 1979, pp. 83–92.
[3] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley, 1998.
[4] Betten Beurs Media, Hagemeyer zet wereldwijde uitrol MOVEX-systeem even stop (Hagemeyer stops worldwide implementation of MOVEX-system). Published via email on June 26, 2003 (in Dutch).
[5] B.W. Boehm, Software engineering, IEEE Transactions on Computers C-25 (1976) 1226–1241.
[6] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.
[7] S.L. Brown, K.M. Eisenhardt, Competing on the Edge—Strategy as Structured Chaos, Harvard Business School Press, 1998.
[8] J.M. Chambers, T.J. Hastie (Eds.), Statistical Models in S, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1992.

[9] Financieel Dagblad, Lot Van Heek-Tweka aan zijden draad (Destiny Van Heek-Tweka hangs by a thin thread), in: Financieel Dagblad, June 20, 2003 (in Dutch).
[10] E.B. Daly, Management of software engineering, IEEE Transactions on Software Engineering SE-3 (3) (1977) 229–242.
[11] G. Dedene, J.-P. De Vreese, Realities of off-shore reengineering, IEEE Software 7 (1) (1995) 35–45.
[12] Department of Defense, C4ISR Architecture Working Group Final Report, 1998, Available via http://www.defenselink.mil/c3i/org/cio/i3/AWG_Digital_Library/pdfdocs/fnlrprt.pdf (Current August 2003).
[13] J.B. Dreger, Function Point Analysis, Prentice Hall, 1989.
[14] J.L. Elshoff, An analysis of some commercial PL/I programs, IEEE Transactions on Software Engineering SE-2 (2) (1976) 113–120.
[15] D. Garmus, D. Herron, Function Point Analysis—Measurement Practices for Successful Software Projects, Addison-Wesley, 2001.
[16] P.R. Garvey, Probability Methods for Cost Uncertainty Analysis—A Systems Engineering Perspective, Marcel Dekker Inc., 2000.
[17] R.L. Glass, Computing Calamities—Lessons Learned From Products, Projects, and Companies that Failed, Prentice Hall, 1998.
[18] R.L. Glass, Software Runaways—Lessons Learned from Massive Software Project Failures, Prentice Hall, 1998.
[19] R.L. Glass, ComputingFailure.com—War Stories from the Electronic Revolution, Prentice Hall, 2001.
[20] M. Hanna, Maintenance burden begging for a remedy, in: Datamation, 1993, pp. 53–63.
[21] C. Harris, Big questions for ICI posed by quest unit, in: Financial Times, 2003.
[22] S. Huet, M.-A. Gruet, Stastical Tools for Nonlinear Regression—A Practical Guide with S-Plus Examples, Springer Verlag, 1996.
[23] S.D. Hunter, Information technology, organizational learning, and the market value of the firm, Journal of Information Technology Theory and Application 5 (1) (2003) 1–28.
[24] C. Jones, Assessment and Control of Software Risks, Prentice-Hall, 1994.
[25] C. Jones, Applied Software Measurement: Assuring Productivity and Quality, 2nd ed., McGraw-Hill, 1996.
[26] C. Jones, Conflict and Litigation Between Software Clients and Developers, 1996, Version 1—March 4. (Technical note).
[27] C. Jones, Patterns of Software Systems Failure and Success, International Thomsom Computer Press, 1996.
[28] C. Jones, Estimating Software Costs, McGraw-Hill, 1998.
[29] C. Jones, The Year 2000 Software Problem—Quantifying the Costs and Assessing the Consequences, Addison-Wesley, 1998.
[30] C. Jones, Software Assessments, Benchmarks, and Best Practices, Information Technology Series, Addison-Wesley, 2000.
[31] C. Jones, Conflict and Litigation Between Software Clients and Developers, 2001, Version 10—April 13. (Technical note).
[32] C. Jones, Personal communication, January 2003.
[33] A.R. Kan, Managing a multi-billion dollar IT budget, in: S.L. Pfleeger, C. Verhoef, H. van Vliet (Eds.), Proceedings of the International Conference on Software Maintenance, ICSM'2003, IEEE Computer Society Press, 2003, p. 2.
[34] R. Kazman, J. Asundi, M. Klein, Making architecture design decisions: An economic approach, Technical Report CMU/SEI-2002-TR-035, Software Engineering Institute, 2002.
[35] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, Technical Report CMU/SEI-98-TR-008, Software Engineering Institute, 1998.
[36] C.F. Kemerer, Reliability of function points measurement—a field experiment, Communications of the ACM 36 (2) (1993) 85–97.
[37] C.F. Kemerer, B.S. Porter, Improving the reliability of function point measurement: an empirical study, IEEE Transactions on Software Engineering SE-18 (11) (1992) 1011–1024.
[38] W. Kobitzsch, D. Rombach, R.L. Feldmann, Outsourcing in India, IEEE Software 12 (2) (2001) 78–86.
[39] A. Krause, M. Olson, Basics of S and S-Plus, 2nd ed., Springer Verlag, 2000.
[40] B.P. Lientz, E.B. Swanson, Software Maintenance Management—A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations, Addison-Wesley, Reading MA, 1980.

[41] S. McConnell, Rapid Development, Microsoft Press, 1996.
[42] A. McCue, Shareholder militancy puts IT outsourcing under microscope—Institutional investors concerned about risks involved in huge deals, 2003, http://www.silicon.com/news/500021-500001/1/5673.html?nl=3Dd20030821 (Current August 2003).
[43] P.V. Norden, Curve fitting for a model of applied research and development scheduling, IBM Journal of Research and Development 2 (3) (1958).
[44] P.V. Norden, Useful tools for project management, in: B.V. Dean (Ed.), Operations Research in Research and Development, Wiley & Sons, 1963.
[45] P.V. Norden, Useful tools for project management, in: M.K. Starr (Ed.), Management of Production, Penguin Books, 1970, pp. 71–101.
[46] J.C. Pinheiro, D.M. Bates, Mixed-Effects Models in S and S-PLUS, Springer Verlag, 2000.
[47] L.H. Putnam, A macro-estimation methodology for software development, in: Proceedings IEEE COMPCON 76 Fall, IEEE Computer Society Press, 1976, pp. 138–143.
[48] L.H. Putnam, A general empirical solution to the macro software sizing and estimating problem, IEEE Transactions on Software Engineering SE-4 (4) (1978) 345–361.
[49] L.H. Putnam, W. Myers, Measures for Excellence—Reliable Software on Time, Within Budget, Yourdon Press Computing Series, 1992.
[50] L.H. Putnam, D.T. Putnam, A data verification of the software fourth power trade-off law, in: Proceedings of the International Society of Parametric Analysts—Sixth Annual Conference, vol. III(I), 1984, pp. 443–471.
[51] L.H. Putnam, R.W. Wolverton, Quantitative management: software cost estimating, in: Proceedings of the IEEE Computer Society First Computer Software and Applications Conference, COMPSAC 77, IEEE Computer Society Press, 1977, pp. 8–11.
[52] J. Reutter, Maintenance is a management problem and a programmer's opportunity, in: A. Orden, M. Evens (Eds.), 1981 National Computer Conference, AFIPS Conference Proceedings, vol. 50, AFIPS Press, Arlington, VA, 1981, pp. 343–347.
[53] B. Roberts, Ratings game, CIO Magazine, 2000, Available via http://www.cio.com/archive/101500_rating.html.
[54] T.D. Steiner, D.B. Teixeira, Technology in Banking—Creating Value and Destroying Profits, Irwin, McGraw-Hill, 1990.
[55] C. Stoermer, F. Bachmann, C. Verhoef, SACAM: The software architecture comparison analysis method, Technical Report CMU/SEI-2003-TR-006, Software Engineering Institute, 2003.
[56] The Standish Group, CHAOS, 1995, Retrievable via http://standishgroup.com/visitor/chaos.htm (Current February 2001).
[57] The Standish Group, CHAOS: A Recipe for Success, 1999, Retrievable via http://www.pm2go.com/sample_research/chaos1998.pdf (Current August 2003).
[58] The Standish Group, EXTREME CHAOS, 2001, Purchase via https://secure.standishgroup.com/reports/reports.php.
[59] United States Government, Joint Industry/Government Parametric Estimating Handbook, 1999, Available via http://www.ispa-cost.org/PEIWeb/finaled.zip (Current August 2003).
[60] United States Government, Sarbanes-Oxley Act of 2002, 2002, Available via http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf (Current August 2003).
[61] W.N. Venables, B.D. Ripley, Modern Applied Statistics with S-PLUS, 3rd ed., Springer Verlag, 1999.
[62] C. Verhoef, Quantitative IT portfolio management, Science of Computer Programming 45 (1) (2002) 1–96.
[63] C. Verhoef, Quantifying the value of IT-investments, Science of Computer Programming (2004) (in press), doi:10.1016/j.scico.2004.08.004, Available via: http://www.cs.vu.nl/~x/val/val.pdf.
[64] E. Yourdon, Death March—The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects, Prentice-Hall, 1997.